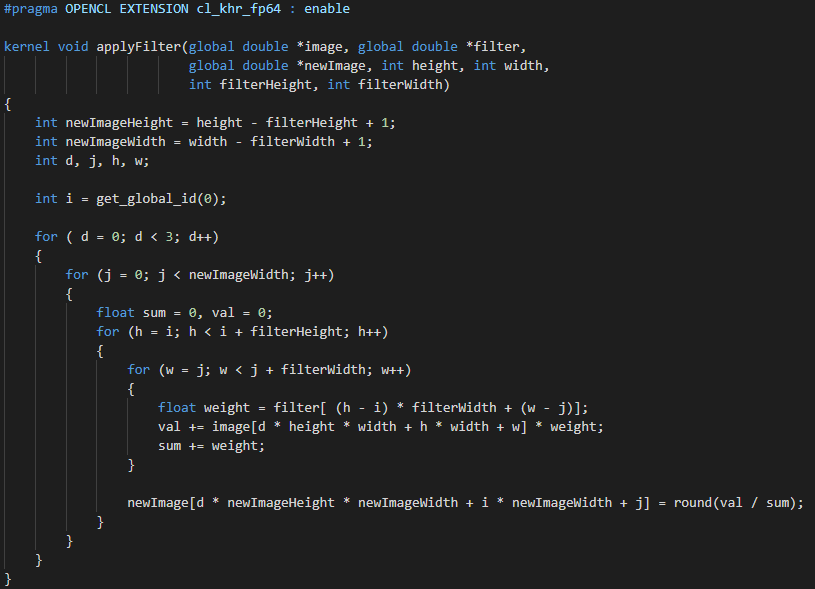


**Práctica 4**

1. Desarrolla una versión en CUDA del código que has desarrollado en la práctica 3 para el procesamiento de imágenes.

Tras un intento fallido con Cuda. Se ha procedido a usar OpenCL de forma satisfactoria. Solamente una función se ha modificado con respecto a la versión anterior. Esta función, es la que vamos a proporcionarle a la gráfica para que realice el filtrado correspondiente. Para poder realizar esto, se ha añadido a un nuevo fichero denominado “clProgram.cl”. De igual forma, se ha cambiado el planteamiento de dicha función para que aplique el filtrado a un vector en específico. Esta función la podremos ver a continuación:

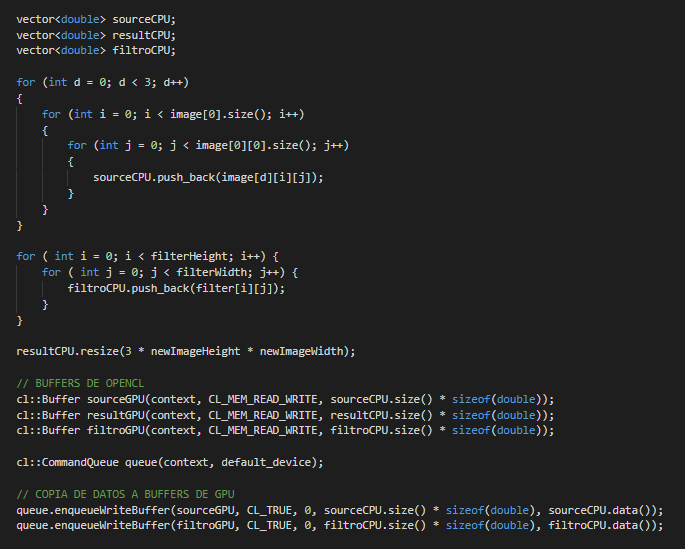


Posteriormente, se ha modificado el “main” del código principal adaptándolo a la sección de OpenCl. Para ello, se han creado las siguientes modificaciones:

**Inicialización de OpenCL**



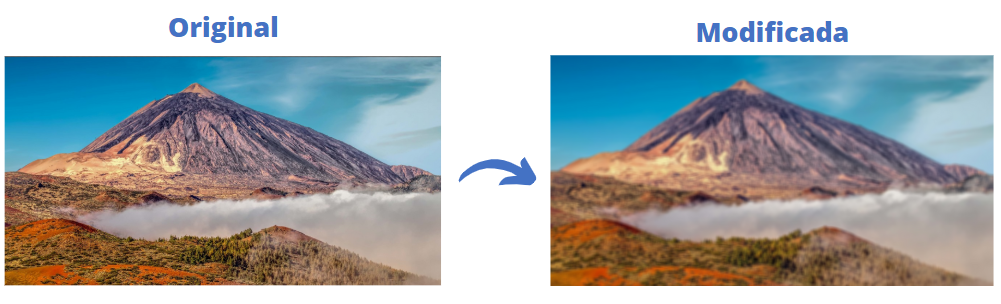
**Preparación de datos en OpenCL**

****

**Ejecución del Kernel**



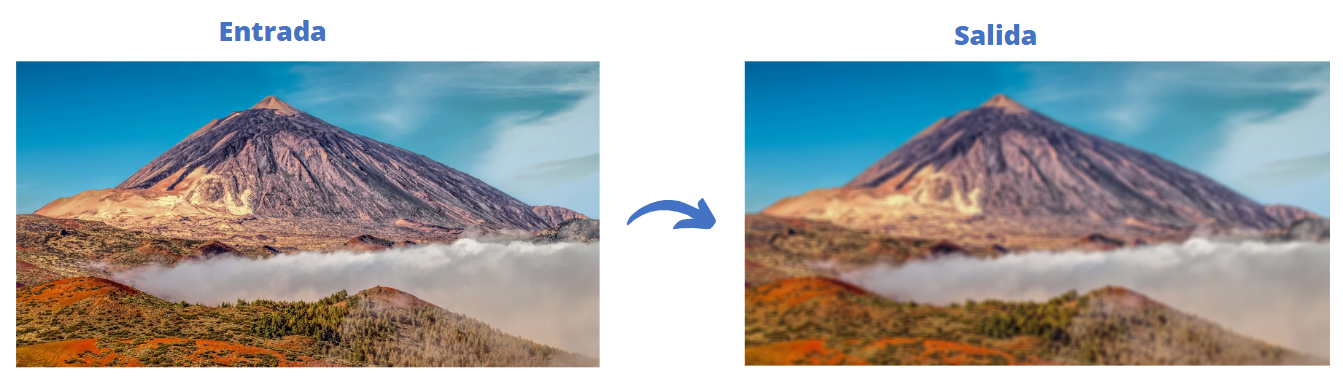
Posteriormente, se ha comprobado el resultado final obteniendo lo siguiente:



2. Analiza el rendimiento de las tres versiones paralelas que has desarrollado.

En primera instancia, podremos ver las imágenes utilizadas y la salida obtenida:

**Imagen Pequeña 1200 x 630**



**Imagen Mediana 2560 x 1440**

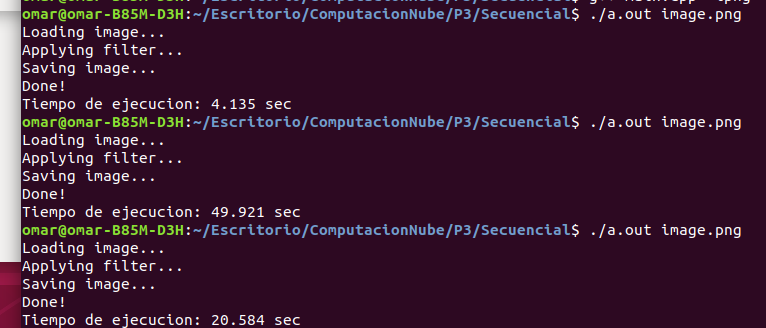


**Imagen Grande 4000 x 2250**



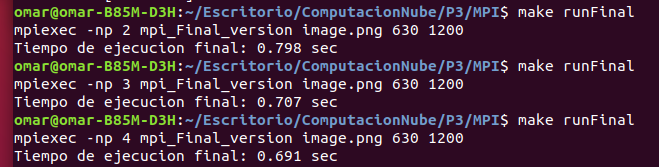
A continuación, podremos ver los resultados obteniendo en la ejecución de los diferentes programas con diferente número de procesadores (si procede):

**Secuencial**

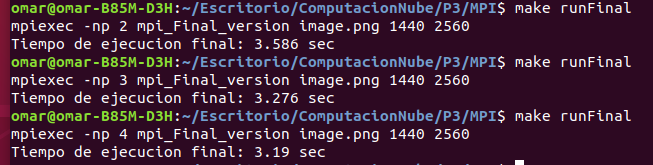
****

**MPI**

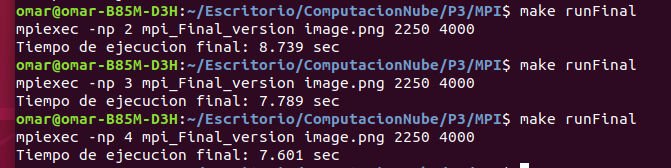
**Imagen Pequeña 1200 x 630**

****

**Imagen Mediana 2560 x 1440**

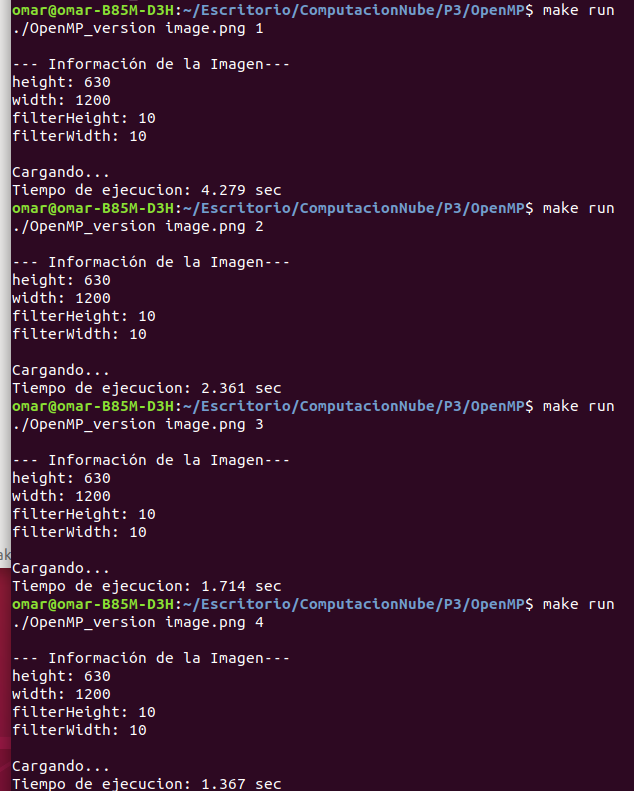
****

**Imagen Grande 4000 x 2250**

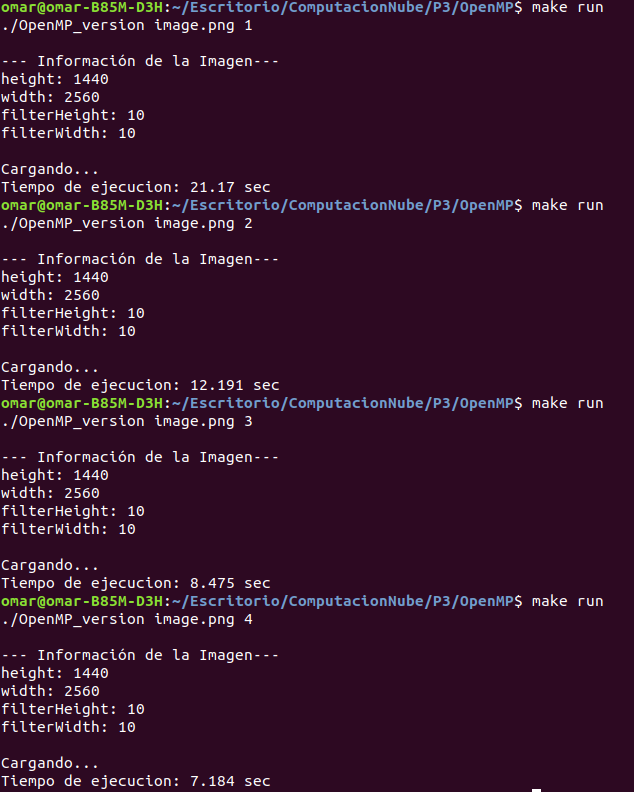
****

**OPENMP**

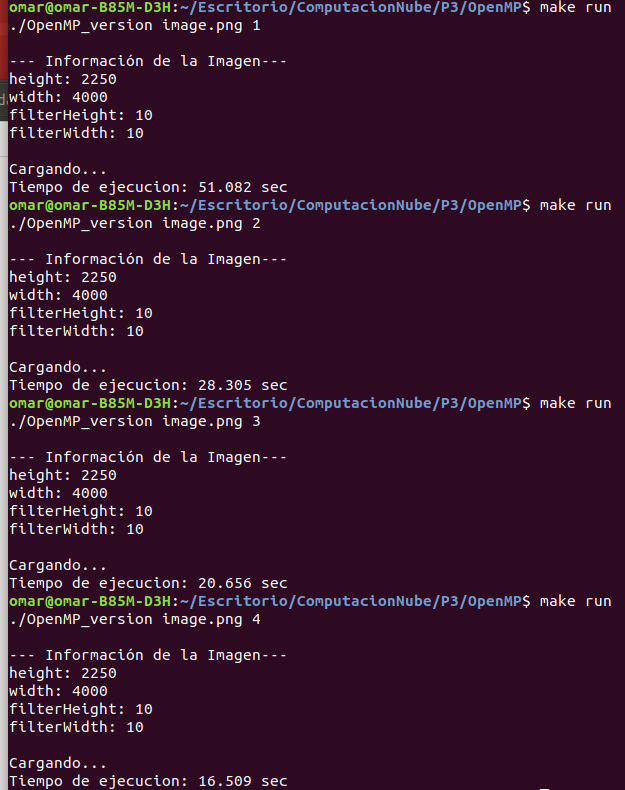
**Imagen Pequeña 1200 x 630**

****

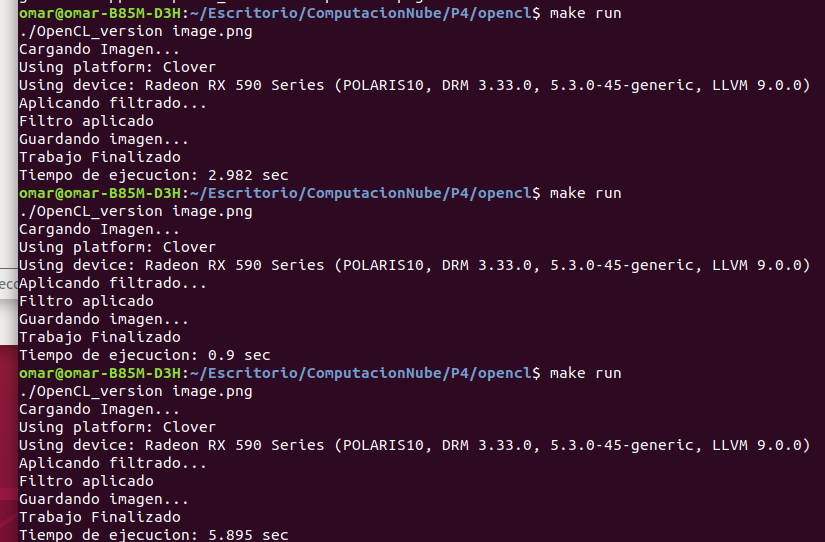
**Imagen Mediana 2560 x 1440**

****

**Imagen Grande 4000 x 2250**

****

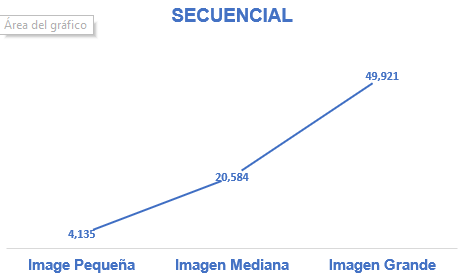
**OPENCL**

****

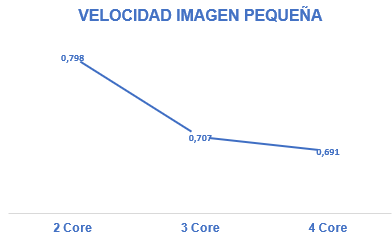
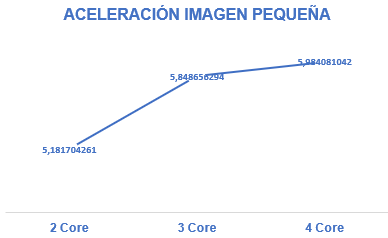
A continuación, podremos ver las gráficas generadas para cada solución propuesta

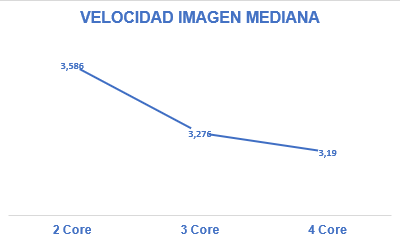
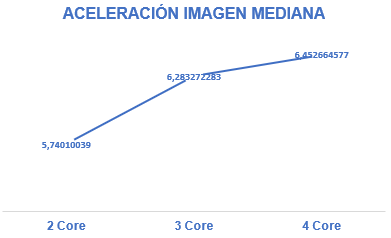
**Secuencial**

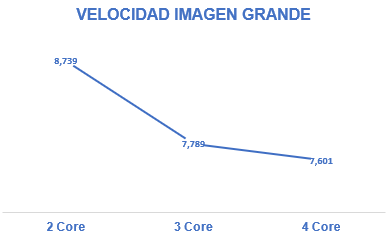
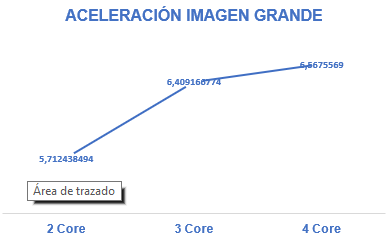
**Gráfica Velocidad**

****

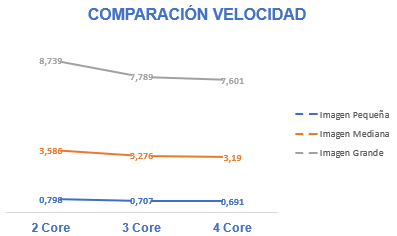
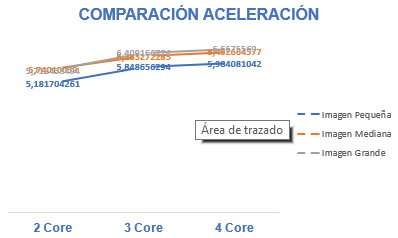
**MPI**

**Imagen Pequeña**

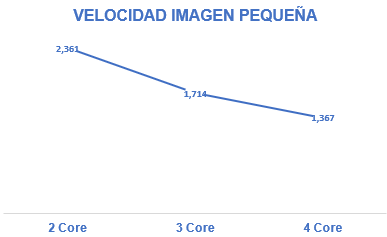
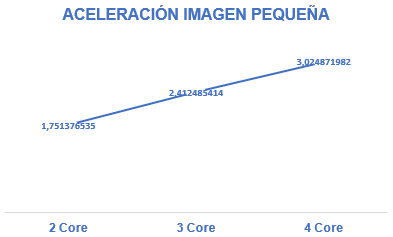
**Imagen Mediana**

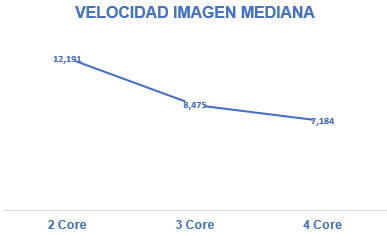
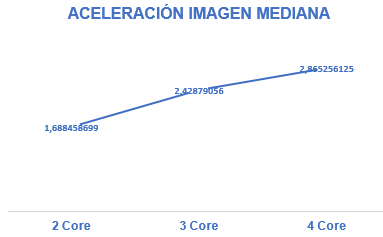
**Imagen Grande**

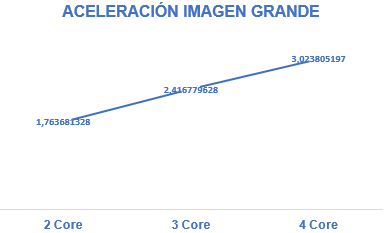
**Resumen**

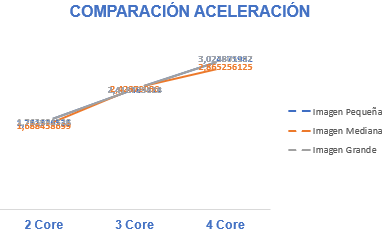
****

**OpenMP**

**Imagen Pequeña**

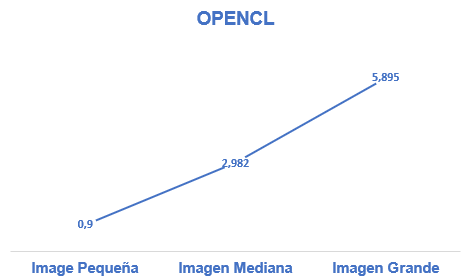
**Imagen Mediana**

**Imagen Grande**

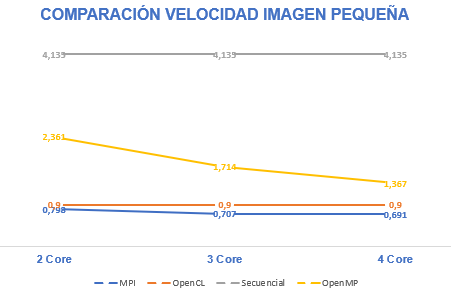
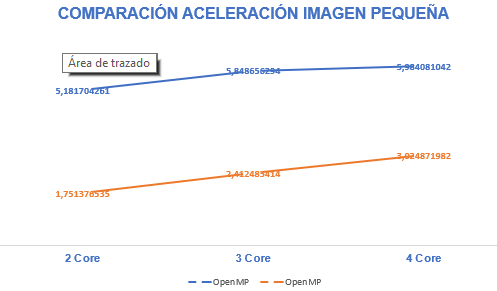
**Resumen**

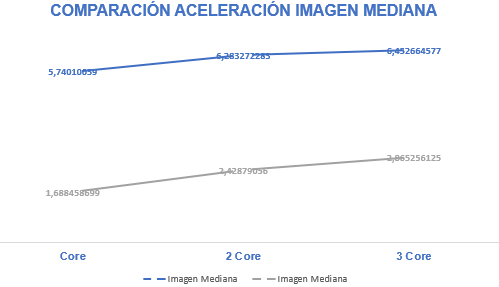
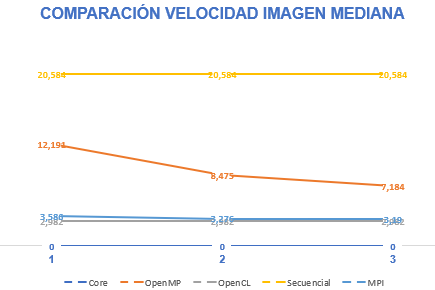
**OpenCL**

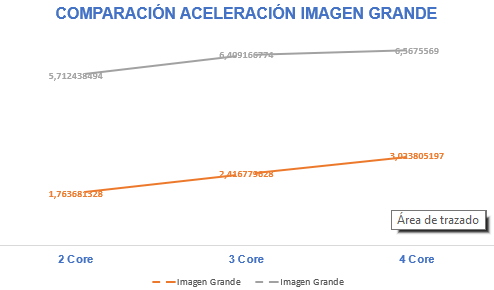
**Gráfica Velocidad**

****

**RESUMEN GENERAL**

** Imagen Pequeña**

****  **Imagen Mediana**

** Imagen Grande**

Las gráficas anteriores se han sacado con un procesador i7 que dispone de 4 cores. De igual forma, se ha usado una gráfica Radeon rx 590 Series. Por este motivo las gráficas llegan a un máximo de 4 cores

Como hemos visto en las gráficas anteriores, la velocidad de cómputo no es directamente proporcional al número de cores. Esto quiere decir que, si podemos ejecutar el código con un Core en 10 segundos, no vamos a poder reducir a la mitad dicho tiempo si ponemos un segundo Core. De igual forma, en la mayoría de los casos la pendiente de la aceleración es menor cada vez que subimos de Core, es decir, el programa se computa más rápido pero cada vez sube menos la aceleración.

Las gráficas anteriores nos permiten comparar las diferentes versiones de forma más sencilla. Gracias a esto podemos decir que, de las versiones creadas la más rápida es la de OpenCL seguida de MPI, OpenMP y secuencial (ordenadas por velocidad). Esto es lógico de forma teórica debido que los códigos paralelizados son más rápidos que los secuenciales. De igual forma, la computación con la gráfica es mucho más rápida para imágenes que el procesador. De la misma manera, es más rápido MPI que OpenMP. Esto es debido a que MPI es más configurable. Sin embargo, OpenMP es más fácil de programar. Por todo ello, opino que si se dispone de una buena gráfica es más rentable (pata este caso) usar OPENCL, Sin embargo, si no se puede usar tendríamos que seleccionar nuestra filosofía, es decir, si se requiere mayor velocidad o simplicidad de programación.